

Memristive Logic in Crossbar Memory Arrays: Variability-Aware Design for Higher Reliability

Manuel Escudero, Ioannis Vourkas, Antonio Rubio, and Francesc Moll

Abstract— The advent of the first TiO₂-based memristor in 2008 revived the scientific interest both from academia and industry for this device technology and has so far led to several emerging applications including logic and in-memory computing. Several memristive logic families have been proposed in the current quest for energy-efficient future computing systems. However, the limited device endurance and variability (both cycle-to-cycle and device-to-device) are important parameters to be considered in the assessment of logic operations. In this work we used an accurate physics-based model of a bipolar memristor (supporting parasitics of the device structure and variability of switching voltages and resistance states) and demonstrate that performance of memristor-based in-memory computations can be degraded owing to both variability and state drift impact, if such features are not properly considered in the design flow. Inspired on pseudo-NMOS ratioed logic and based upon a CMOS-like logic scheme, we propose a crossbar-compatible memristive ratioed logic style which is tolerant to device variability and does not affect device endurance as computations do not involve conditional switching of memristors. Using the Cadence Virtuoso suite, we compare this logic scheme with MAGIC and CNIMP approaches, focusing on the universal NOR gate and complex logic functions.

Index Terms—memristor, MAGIC, CNIMP, IMPLY, ratioed logic, in-memory computing, resistive switching, crossbar array

I. INTRODUCTION

THE existence of the memristor as the fourth fundamental circuit element was postulated by Leon Chua in 1971 [1]. However, this emerging device technology drew unprecedented attention quite later, after 2008, when a group at Hewlett-Packard Laboratories (HP Labs) demonstrated the first TiO₂-based memristor [2], connecting the nature of such devices with Chua's previous theory. Owing to their analog nature, potential nonvolatility, high integration density and CMOS BEOL compatibility, memristors constitute an important trend in modern electronics [3], representing a very promising technology which has extended its influence beyond memory [4] to logic [5], neuromorphic and *in-memory* computing [6], [7].

The 2008 HP Labs invention also concerned the development of a simplistic device model, which has been ever since the basis for several more models published later [8-10]. The majority of such models capture the very basic behavioral device characteristics (e.g. threshold-based switching and nonlinearities near the resistive boundaries applying window functions [11], to name a few), and thus contributed to the progress of this emerging research field, being adequate to demonstrate the impact and usefulness of memristors in a variety of applications [12], [13]. However, recent advanced physics-based device models [14] go deeper into the device dynamics and take into account parasitics (owing to the device structure), variability of voltage thresholds and resistance states, temperature dependencies, cycle-to-cycle stochasticity, features normally observed in real device implementations, enabling consequently more realistic circuit simulations.

Among several potential applications, the memristive logic design, i.e. the methodology of designing logic circuits using memristors, is an emerging concept in the constant quest for energy efficient post-CMOS computing systems. Many such memristive logic families have been proposed: IMPLY [15], MAGIC [16], CNIMP [17], MRL [18], MAD [19], to name a few, in which resistance is normally used to represent data, thus all are suitable for resistive computing, most of them also being compatible with the crossbar geometry. The latter is considered a requirement for real *in-/near-memory* computing, since the topology of the logic circuits to implement should fit in the crossbar memory array, which is most likely the prevailing topology for memristive memory. What is more, the metrics proposed so far for the comparison of such logic families naturally focus on latency, energy, and area efficiency [20]. However, most such relevant works omit crucial factors such as variability (both cycle-to-cycle and device-to-device) and endurance of memristors; being the latter a major limitation to be considered if frequent switching is necessary during computations. For instance, recently Papandroulidakis *et al.* [21] suggested using different

memristor technology for memory and computing tasks to address endurance issues. Likewise, another logic scheme called Scouting Logic [22] was proposed to alleviate the endurance requirement while executing logic operations by just sensing the memristor state, even though this scheme eventually suffers from device variability. Moreover, MAD gates [19] were proposed based on the same principle (using Memristors As Drivers) but the target topology was not compatible with the crossbar geometry. Performing logic computations without switching the states of the involved memristors was first seen in the so called CMOS-like memristive logic [26], revisited recently in [27], which was unfortunately not crossbar-compatible either.

In this work, we build upon ideas commented in [24], [25] and present a crossbar-compatible ratioed logic scheme which is completely variation-tolerant; i.e. it does not suffer from device variability or state drift, and also it does not impact the device endurance as logic operations are performed by just reading the state of memristors acting as inputs. We performed circuit simulations of the proposed scheme using the Cadence Virtuoso suite and compared its performance with the two most popular memristive logic families of the literature (MAGIC and CNIMP), focusing on the universal NOR gate and on more complex logic functions such as AND-OR-Invert (AOI). We based our simulations on an advanced physics-based model of a bipolar metal-oxide resistive RAM (ReRAM) device [14], [23] derived by Stanford and our results underline the importance of taking into account device variability in circuit simulations as both read and write errors can emerge due to variability and state drift impact. Such features are rarely considered so far in other relevant publications in this field and the presented results highlight that variability can be critical for proper device selection and circuit design quality/viability assessment. Finally, we proved that the new proposed logic scheme outperforms the rest in terms of robustness, viability and also scalability (*fan-in*), getting us to the simple conclusion that “rethinking of memristive logic design from a practical point of view” is necessary if we aspire to enable *in-memory* computing.

II. TARGET MEMRISTIVE LOGIC GATE DESIGN STYLES

This section briefly describes the three different crossbar-compatible logic gate design schemes whose in-crossbar operation is later studied and compared in the presence of device variability (MAGIC, CNIMP, Ratioed Logic). The same logic function is always performed (NOR $_n$; i.e. a NOR gate with n inputs) and in all cases it is assumed that a memristor stores a logic ‘0’ when it is at a high resistive state (HRS) and logic ‘1’ when it is at a low resistive state (LRS). Hereinafter we will refer to a memristor being forward/reverse-biased when the voltage at the top/bottom terminal is higher than that on its bottom/top terminal; the bottom terminal is denoted by the thick black line in the circuit schematic (See Fig. 1a). So, a SET process (HRS \rightarrow LRS) occurs when the device is forward biased with a voltage of amplitude higher than V_{SET} threshold, whereas a RESET process (LRS \rightarrow HRS) occurs when it is reverse-biased with a voltage amplitude higher than $|V_{RESET}|$ threshold.

A. Memristor-Aided loGIC (MAGIC)

According to Memristor-Aided loGIC (MAGIC) proposed in [16], different logic gates are built using networks of memristors and computation takes place by applying voltage pulses to the whole network. Each logic gate is composed of one or more input memristors, holding the input logic values in their resistive states, and a single output memristor to store the result of the logic operation. It is worth mentioning that only NOR MAGIC gates are crossbar-compatible, thus any logic function shall be computed using sequential NOR operations. More specifically, Fig. 1b shows a generalized topology for a NOR $_n$ gate implemented in a row of a crossbar array. The input data participating in the operation ($x_1 \dots x_n$) is stored in the resistive state of a set of input memristors, whereas an additional output memristor eventually stores the result of the logic computation. Assuming that the input data is initially stored in the memory array, MAGIC NOR $_n$ is evaluated in two steps, regardless of the number of inputs: first, the memristor storing y (output data) is SET to LRS. Next, input voltage pulses of amplitude $V_0 > 0$ are applied to the top terminal of every input memristor and the top electrode of the output memristor is connected to ground. Note that at the same time the row line must show high impedance (floating) in both ends. Thus, the gate consists of a pull-up network of input memristors and a single pull-down output memristor. The latter is conditionally RESET to HRS only if at least one input memristor is at LRS. V_0 value is selected such that guarantees the switching of the output memristor while being non-destructive for the input memristor states. So its value depends both on the memristor parameters as well as on fan in. For example, according to [16], for a 2-input NOR while assuming a very high R_{HRS}/R_{LRS} ratio where R_{HRS} and R_{LRS} are the memristor resistances in HRS and LRS respectively, the input voltage requirements are as follows: $2 \times |V_{RESET}| < V_0 < V_{SET}$. In other words, MAGIC NOR gates could not be implemented unless $2 \times |V_{RESET}| < V_{SET}$ is true in a particular device technology.

B. Converse NonIMplication (CNIMP)

Converse NonIMplication (CNIMP) was proposed in [17] and concerns a logic gate design style that improves the well-known IMPLY logic [15] by solving the important issue of incomplete switching of the memristors; the latter was originally stated in [28] and was also observed experimentally [29]. The CNIMP and the SET programming operations together form a universal logic gate set. CNIMP was named after the binary operation it reproduces using a circuit consisting of memristors and an auxiliary resistor R_G , able to fit in a row of a crossbar array as shown in Fig. 1c. CNIMP operation is performed by applying voltage pulses of amplitude $V_{COND+} > 0$ and $V_{COND-} < 0$ to the memristors containing the first and second operand, respectively. Figure 1c shows how to realize a NOR2 operation in 3 steps: first the device to finally hold the output data is programmed to LRS. Next, the operation x_1 CNIMP y is performed (intermediate result stored in y) and finally the x_2 CNIMP y to store the result of NOR2 (x_1, x_2) in y . The values V_{COND+} , V_{COND-} and R_G are selected to guarantee: (i) no state change in the memristor corresponding to the first operand, and (ii) conditional change in the state of the second operand to hold the output data. Similar to the MAGIC case, CNIMP involves conditional switching, which is why the requirements for the applied voltage pulse amplitudes are very similar: $2 \times |V_{RESET}| < (V_{COND+} - V_{COND-}) < V_{SET}$ when $R_{HRS} \ll R_G \ll R_{LRS}$ [17]. Therefore, unless $2 \times |V_{RESET}| < V_{SET}$ is true in a particular device technology, CNIMP gates cannot be implemented.

C. Ratioed Logic with Memristors

This target logic style concerns a newly proposed memristor-ratioed logic scheme, inspired on the pseudo-NMOS logic gate design, that it is the key style investigated in this paper. It works very similar to the CMOS-like logic style [26] but it involves much less circuit complexity, making it compatible with crossbar arrays. Two different schemes are shown in Fig. 1c, which include a PMOS (NMOS) transistor connected to a pull-down (pull-up) memristor network representing a logic function. These memristors substitute the conventional NMOS pull-down block or PMOS pull-up block of a CMOS gate, e.g. a block of NMOS transistors in parallel represents a NOR n . Memristors hold input data; a memristor in LRS substitutes a turned ON transistor, whereas a memristor in HRS substitutes a turned OFF transistor. As we decided by convention in which HRS represents a '0' and LRS represents a '1', this is equivalent to a NMOS transistor states when a '0' or a '1' is applied. However, PMOS will be substituted with complemented inputs, i.e. a PMOS with applied '0' ('1') is equivalent to a LRS (HRS) but this state represents a '1' ('0') when stored in a memristor. For instance, a block of PMOS transistors in parallel represents a NAND n but using complemented inputs it turns an OR n . Anyway, the operation is executed by applying a voltage through all the circuit equal to V_{read} , making sure that the voltage across the whole memristor network is appropriate for just "sensing" the input memristor states without disturbing them. Contrary to MAGIC and CNIMP, the logic output corresponds to a voltage level V_{out} whose value falls between $V_{REF} = V_{DD} - V_{read}$ and V_{DD} . The overall circuit in the computing operation works as a resistive divider, delivering the output voltage described in (1), where R_{eq} is the whole memristor network equivalent resistance and R_L is the PMOS channel resistance. Finally, V_{out} is compared to a threshold value V_{COMP} , properly selected so that the output voltage levels are distinguished between '0' and '1'.

$$V_{out} = V_{REF} + (V_{DD} - V_{REF}) \frac{R_{eq}}{R_{DS} + R_{eq}} \quad (1)$$

Figure 1d depicts a ratioed NOR n logic gate circuit in a crossbar row surrounded with a simplified peripheral circuitry to understand the whole operation. The logic operation is performed when write/op signal is '0'. Pull-up PMOS transistor is tuned to exhibit a channel resistance $R_L \approx R_{LRS}$. Ratioed circuits depend on proper pull-up/down resistance for correct operation. In this case, once V_{REF} and V_{DD} pulses are applied (enable signals en1 ... enn select which memristors are working), two different cases are observed depending on the states of the input memristors. For instance, in the NOR2 example in Fig. 1c, when input is "00" the input memristances are both R_{HRS} and V_{out} is roughly V_{DD} , which is interpreted as logic '1'. However, if at least one input is '1' then $V_{out} < [(V_{DD} - V_{REF})/2 + V_{REF}]$ covering the range of logic '0'. Then, the resulting V_{out} is compared to a threshold value V_{COMP} between V_{DD} and, properly selected so that the output voltage levels for '0' and '1' are clearly distinguished. The details of this whole operation are listed in Table 1 using (1). Therefore, the result of computation is not simultaneously stored in the memory array; in this sense, it is reasonable to talk about *near-memory* computing. However, the result can be stored back to a memory element right afterwards as a simple additional programming step. This is done first by storing momentarily the output using the flip-flop. Then, when write/op = 1, the output is written back in a

memristor, by means of the enable signals. The flip-flop and the comparator will add some area and power overhead indeed, but the rest of the peripheral circuitry is also needed for the other logic styles. It is worth mentioning that avoiding the switching of memristors during logic computations is beneficial in many different aspects, and the advantages of the proposed ratioed logic scheme are further demonstrated in the rest of this work.

Although it is not the main scope of this study, it's appropriate to discuss the compactness of this logic scheme in front of the former ones. MAGIC and CNIMP may perform row-wise and column-wise operations and parallel in a 1R crossbar array, using isolation voltages in order not disturb the rest of the memristors. In the case of memristor ratioed logic, a 1T1R crossbar is preferred, because the operation is sensing a voltage and the rest of the crossbar may influence the reading. Assuming a 1T1R crossbar where selector transistors are connected in rows, NOR operations can be performed in row-wise or column-wise, but only parallel operations are column-wise allowed, as shown in Fig. 2. However, memristor-based ratioed logic allows several modifications to accommodate other useful functions. First one, OR_n gates are possible just including an inverter stage after the comparator or using a parallel memristor pull-up block as explained before. With OR_n gate directly available, a particular case for $n = 1$ is the COPY gate, allowing to put the content of a cell in another one in just one step, unlike MAGIC and CNIMP which need two steps (two NOT gates). Additionally, in the NOR2 gate V_{COMP} can be adjusted to fall between '01' and '11' V_{out} value (see Table I). This allows us to perform an AND gate. Note that for higher number of inputs, these two V_{out} values get too close, being incapable of distinguish them correctly. Just by assuming that NOR, OR, NOT and COPY are available, Table 2 shows the cross-point area needed to compute a NOR, OR, AND or NAND for each logic style, where memristor-based ratioed logic wins in this field.

Finally, we would like to comment some issues about scalability issues in terms of logic realization for these logic styles. CNIMP (or IMPLY) and MAGIC mapping has been covered in some works, e.g. in [33] and [34]. They rely on the capability of parallel row and column-wise operations of NOR and NOT gates and propose optimizing algorithms for faster, less consuming area and power consuming logic realizations. Memristive ratioed logic style performs row and column-wise operations, but only one of them is allowed to perform in parallel. However, this logic style can allocate the output in any place and COPY operation only takes one step instead of two steps as MAGIC and CNIMP do. Hence, memristor-based ratioed logic has potential to allow efficient mapping, but using different strategies than MAGIC or CNIMP.

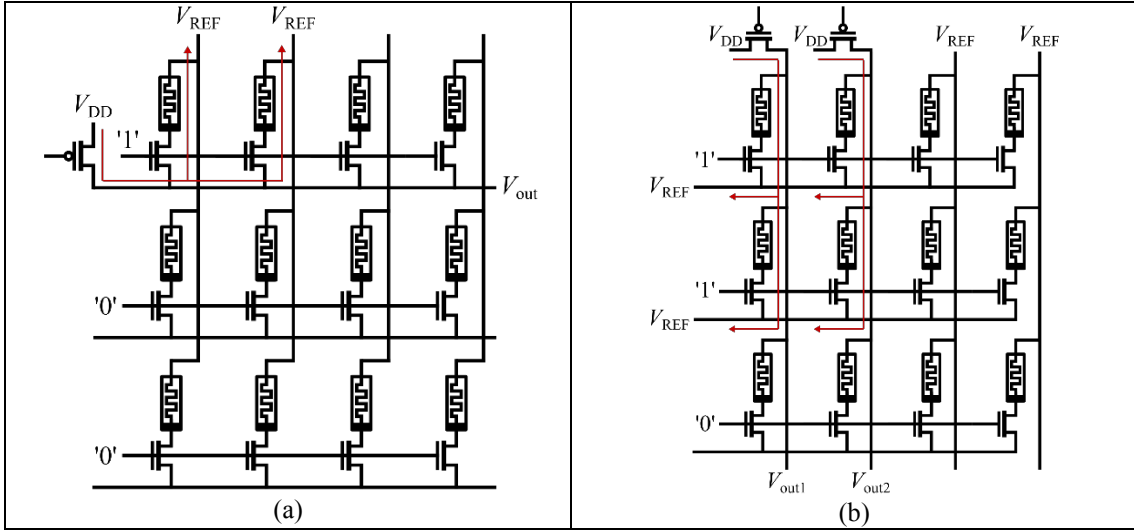


Fig. 2. Possible settings to perform a memristor-based ratioed logic NOR_n in a 1T1R crossbar. (a) Row-wise NOR2, (b) Two column-wise NOR2 in parallel delivering V_{out1} and V_{out2} outputs. Voltage pulses, working PMOS and selected rows are indicated.

III. VARIABILITY-AWARE PHYSICS-BASED MEMRISTOR DEVICE MODEL FOR REALISTIC SIMULATION

A. Description of the Memristor Model

In order to perform more realistic circuit simulations, advanced physics-based models that capture most of the device behavioral characteristics are required. Here we briefly provide the basics of the memristor model used in this work, which is the Stanford-PKU ReRAM model [14]. It is a compact physics-based

model (described in *Verilog-A*) which captures typical DC and AC electrical behavior of metal-oxide based ReRAM devices. The model assumes a conductive filament (CF) growth process described by a change of the CF geometry during the SET and RESET processes under various bias conditions. Thus the core of the model is a two-dimensional description of a unique CF, which includes both the CF gap region and the CF width as control variables. Most importantly, the model includes parasitic effects such as the parasitic resistance of the switching layer and the electrodes, as well as the parasitic metal-insulator-metal (MIM) capacitance. Furthermore, the model supports intrinsic variation effects, such as statistical distribution of switching thresholds and resistance states, temperature dependency and dynamic current fluctuations for the RESET process, thus supporting literally all the ReRAM device variation effects known to date. Operation is very similar to that of other models. A positive applied voltage produces a SET process, where the oxide layer suffers a soft-breakdown; the CF is formed and the device is found at a low resistive state. On the contrary, a negative applied voltage causes a RESET process in which the CF is dissolved through ion diffusion or drift processes.

The model is executed in the Cadence Virtuoso suite. The majority of the parameter values were kept at default values as recommended in [23], except those directly affecting the switching thresholds, i.e. the average active energy of oxygen vacancies (E_a), the hopping barrier of O_2^- (E_h) and the energy barrier between the electrode and the oxide (E_i). Tuning these parameters is recommended to adjust the overall device behavior according to the target application requirements. For instance, in order to comply with the previously mentioned threshold voltage requirements for MAGIC and CNIMP logic operations ($V_{SET} > 2 \times |V_{RESET}|$), these parameters were tuned as follows: $E_a = 0.9$ eV, $E_h = 0.9$ eV and $E_i = 0.7$ eV.

Figure 2a demonstrates i - v curves for 20 cycles taken for a device under a triangular voltage application. The compliance current (cc) is adjusted by tuning the gate voltage of a series NMOS transistor (we use here realistic $0.35 \mu m$ MOS models). Since the memristor model does not include *voltage thresholds* directly as parameters, we define the SET threshold V_{SET} the voltage at which the current reaches to 90% of the cc . Likewise, we define as RESET threshold V_{RESET} the voltage when the current first experiences a sudden decrease. These statistics give us the following mean values: $V_{SET} = 2$ V and $V_{RESET} = -0.5$ V, which will allow us later to establish the appropriate amplitude of the programming/reading voltage pulses. Moreover, the conduction of the model is purely ohmic for the conductive filament body (LRS) but non-linear for the tunneling gap (HRS). This is observed in Fig. 2b which shows how the effective R_{HRS}/R_{LRS} ratio can change depending on the applied voltage. The device is first SET to R_{MIN} , the lowest resistive value, and then a positive voltage is applied without modifying its state. As expected, R_{LRS} shows the ohmic conduction of the CF. Nevertheless, when the device is RESET to R_{MAX} , the highest resistive value, and a larger negative voltage is applied, again without disturbing its state, interestingly a highly nonlinear behavior is observed in the effective R_{HRS} owing to the hopping current through the tunneling gap. Therefore, such dependency of the effective R_{HRS} state on the voltage across the device marks a significant difference when compared to other device models commonly used in memristive logic design. This fact could have a significant impact on the efficiency of memristive applications, thus it is an important consideration also for logic gate design process.

B. Memristor State Encoding, Read Out, and Variability

Binary encoding of memristance is necessary for logic applications. Given the used parameter values, the model exhibits a memristance range from $5 k\Omega$ (R_{MIN}) to $3 M\Omega$ (R_{MAX}). Taking into consideration the results in Fig. 2b, we defined values above $1 M\Omega$ as HRS and values below $100 k\Omega$ as LRS, whereas all values within the intermediate guard band are treated as undefined states (see memristance map in Fig. 3a). Reading of the memristor state is performed in voltage mode with a $100 k\Omega$ series resistor by applying a small voltage pulse ($V_{read} = 0.5$ V, 20 ns- wide) which does not affect the device state. For the purposes of our simulations, we implemented the state decoder shown in Fig. 3a (adapted from [30]). The voltage read through the memristive voltage divider is compared with two reference values that represent the upper bound of LRS, i.e. $R_{LRS,h} = 100 k\Omega$, and the lower bound of HRS, i.e. $R_{HRS,l} = 1 M\Omega$. So, depending on the result of the comparison, the output of the circuit is either logic ‘1’ for LRS, logic ‘0’ for HRS, or ‘X’ for any undefined intermediate resistive state.

An important aspect of the Stanford/PKU model used in this work, is the capability of generate cycle-to-cycle variability. During the switching process, a random variable is added to the change rate of the tunneling gap distance between the electrode and the tip of the conductive filament (CF) g , and to the change rate of the CF width w . Such random variable is a zero-mean Gaussian distribution with std. deviation σ_g and σ_w , respectively, thus modifying g and w in every step of the transient simulation. Different devices shown different amounts of variability, hence it is interesting to include different scales of statistic distributions for these parameters. To do so, we set $\sigma_g = k \times \sigma_{g0}$ and $\sigma_w = k \times \sigma_{w0}$ ($\sigma_{g0} = 10^{-4}$ m/s and $\sigma_{w0} =$

$5 \cdot 10^{-4}$ m/s are the default values), where $k = 1, 2, 3 \dots$ is for us a variability factor that permits configuring easily the amount of desired variability to satisfy the required R_{LRS} and R_{HRS} distributions. Such state variability affects the memristance values as well as the switching thresholds (as shown previously in Fig. 2a applying $k=1$).

We inject variability to the initial states of the devices through a two-step initialization process, as shown in Fig. 3b: when programming the device to the LRS (HRS), this is done by first performing a hard RESET (SET) and then a soft SET (RESET). Hard SET (RESET) completely forms (destroys) the CF to thus eliminate the previous history of the memristor. On the contrary, the HRS or LRS programming taking place right next (called soft programming) initializes the memristor to a state within the LRS or HRS range, as desired, thus including the variability effect in the memristance. The voltage pulses applied for the HRS initialization concern: V_{ON} amplitude of 3 V, 200 ns width and 500 μ A cc for hard SET, $V_{OFF,soft}$ amplitude of -2 V, and 100 ns width for soft RESET. On the other hand, for the LRS initialization it is: $V_{OFF,hard}$ amplitude of -2.5 V, 200 ns width for hard RESET, and V_{ON} amplitude, 100 ns width and 50 μ A cc for soft SET. Figure 3c shows simulation results for the read memristance distribution of R_{LRS} and R_{HRS} after such initialization process. It can be observed that variability-aware simulations for HRS/LRS programming show normal distributions for the device state whose std. dev. increases linearly with increasing values of the variability factor k .

IV. TARGET SYSTEM OVERVIEW: CROSSBAR MEMORY ARRAY AND PERIPHERAL CIRCUITRY

Crossbar is the target memory topology in this work and all considered logic design schemes are crossbar-compatible. Therefore, for the purposes of our simulations we designed both the crossbar memory array as well as the entire peripheral circuitry used to properly program the devices, read their states, and perform the logic operations of interest according to the logic styles described previously.

The main unit of the simulated system is the memristor crossbar array shown in Fig. 4a, where each cross-point cell consists of a memristor and a series select transistor (1T1R). 1T1R arrays are known to avoid unwanted sneak-path currents and isolate the working memristors from the rest of the crossbar. The drawback is a penalty on packing density, that it can be near 2.5 times the cell area as investigated in [31]. However, it may be compensated by the improvement in reliability. We assume that the memristors are stacked directly on top of group-accessed select transistors, which are placed on the bottom layer (fabricated as front-end transistors) to limit the influence of the parasitic capacitance and resistance and also to minimize the area penalty (such as in the case of [32] with the nano-pillar gate-all-around transistors).

Moreover, there are multiplexer (MUX) and demultiplexer (DEMUX) units to drive the selected columns and rows. The MUX, whose detailed implementation is shown in Fig. 4b, provides different voltage levels to the rows or columns. It is primarily composed by a decoder, which connects the output to any of the input voltages according to the selection signals sel_{mux} , or disconnects completely the block from the crossbar (high impedance node - HZ) using the enable signal en . The input voltage signals are shown in two groups; those concerning memory operations and those used in logic operations with either of the three logic styles supported here. On the other hand, the DEMUX shown in Fig. 4c, provides connection to extra circuitry when necessary, such as the reading circuit described in Fig. 3a, the transistor that limits current in the LRS or HRS programming and the R_G used in the CNIMP logic style. There are also pull-up PMOS transistors connected to both rows and columns, which are enabled only when computations using the memristor-based ratioed logic take place (see Fig. 1c).

It is worth noting here that the system was designed particularly to support all features necessary in the three different logic styles (apart from the typical memory operations.) So, the area overhead of the peripheral circuitry is not an issue here since it will be severely reduced if just one logic style is desired.

V. SIMULATION RESULTS AND DISCUSSION

Next we show how the setup of the target system is implemented to simulate the different in-memory operations, which allow us to eventually evaluate the impact of device variability in the design process of memristive logic gates.

A. Simulation Setup

The system described in the previous section was implemented in the Cadence Virtuoso design environment. For the peripheral blocks, we used Verilog-A behavioral description of decoders and comparators, whereas ideal switches were used for the transistors to minimize simulation overhead and focus on the parasitics owing to the 1T1R topology. Therefore, the cross-point select transistors and those

directly participating in the programming functions and in the memristor-ratioed logic scheme were implemented using 0.35 μm standard CMOS technology. The inputs of the simulation were primarily all the voltage pulses applied to MUXes, DEMUXes and the pull-up transistors that enable each operation. The target logic functions are the NOR2, NOR4 and a 2-2 AND-OR-Invert (AOI22). Among them, the NOR2 provides an easy to understand design and straightforward variability analysis, whereas with NOR4 the increment of fan-in is explored. Eventually, the AOI22 is a complex function that requires chained operations, which was built as a two-level NOR2 implementation using inverted inputs. For the chained operations, two auxiliary memristors were employed to hold the output of the first level NOR2 gates and serve as input for the last level NOR2 gate.

In the early design steps, the main goal was to find the design parameters for each operation that guarantees the expected outcome, also called the *design space*. This in MAGIC primarily concerns selecting V_0 , whereas in CNIMP it is about $V_{\text{COND+}}$, $V_{\text{COND-}}$ and R_G . Finally, for the proposed ratioed logic the load transistor needs to be properly designed and the V_{DD} , V_{REF} values to be properly selected. The method followed to find the proper design space was to sweep the design parameter values for every possible combination of the input variables of the logic operations. Variability impact was evaluated considering the error rate as primary metric, which we compute as the number of unsuccessful logic operations over the total number of logic operations performed. The latter is valid as long as the state of both the input and output memristors is not driven to any undesired region after a logic operation is performed. The error rate is computed using 1000 simulations for each input combination of interest. MAGIC and ratioed logic NOR n are computed as a commutative operation, i.e. NOR2 input cases “01” and “10” are equivalent, hence only “01” was considered. However, CNIMP NOR n was computed via a sequence of CNIMP operations, thus “01” and “10” are by default different operations even though the logic result is expected to be the same, so both cases were simulated. Finally, the average error rate is calculated, which for MAGIC and ratioed logic styles it is a weighted average error rate from the simulated input cases (e.g., in NOR2 “00” and “11” have a weight of 0.25, whereas “01” has a weight of 0.5 since it includes also the equivalent “10” case.)

In order to accommodate large number of operations with several memristors and to simplify the configuration of every simulation, a Python script was prepared which works similarly to a compiler program, as follows: it takes as inputs the *crossbar size* and an *instruction file* that includes the list of operations to be performed in the system during the simulation and generates all the input signals required (e.g., enable and select signals for MUX/DEMUX blocks, gate signals for the row select transistors and the PMOS devices, etc.) in the form of piece-wise descriptions w.r.t. time, as required by the Cadence simulation environment. The instruction file is a sequential list of all the different possible operations supported (i.e. HRS and LRS programming, READ, MAGIC NOR, CNIMP and RATIOED NOR) along with several arguments related to the array position of memristors involved in the operation or specific features for certain operations, such as the number of inputs of a logic gate, etc. The whole simulation flow is shown in Fig. 5a. An example of a simulation file for a MAGIC NOR2 with previous inputs programmed at “01” and memristors read before and after the simulation, is shown in Fig. 5b. Once the piece-wise linear files are generated by your compiler, the Cadence Tool incorporates them along with the schematics, device models, variable values, and the simulation is executed for the given scenario. Finally, the simulation results are exported, containing the state of all the memristors of interest, both before and after the execution of the logic operation(s). It is worth mentioning that such compiler is versatile enough to permit being extended in order to support more logic styles (by updating the peripheral circuitry blocks,) as well as optimized for less styles, while incorporating different models of memristors and transistors.

B. Simulation Results

1) MAGIC logic gates

Regarding MAGIC, the proper design space for NOR2 and NOR4 was first explored and the impact of variability was then evaluated for both, including the AOI22 function. Note that AOI22 does not require to be designed separately as it is composed of sequential NOR2 operations. In the design process, the V_0 value was swept while monitoring the memristance of all the memristors after the logic operation. The V_0 voltage pulse width was kept at 200 ns, time long enough to perform the switching of the output memristor. If any input memristor did not hold its initial logic level, or if the logic level of the output memristor was not the expected one, then the specific V_0 value was not included in the design space. This process took place for every distinct input combination (“00”, “01/10” and “11”).

Figure 6a presents the results for NOR2, i.e. the memristance of the input and output memristors after the logic operation. In the x-axis the valid V_0 range is shown, extending from 1.89 V to 2.19 V. The initial HRS ($R_{\text{HRS,ini}}$) and LRS ($R_{\text{LRS,ini}}$) is indicated by a dashed black line. The higher bound of the V_0 range is found in the “00” input case as the state of the input memristors enter the undefined region for $V_0 > 2.19\text{V}$.

Likewise, the lower bound is found in the “11” input case, when the state of the output memristor enters the undefined region when $V_0 < 1.89\text{V}$. Note that in the “00” input case the input memristors do not switch their state from HRS to LRS; however, we notice a state drift (see Fig. 2b) towards the undefined region, due to the considerable voltage drop on the input memristors, which is insufficient to cause a complete switching event but it is high enough to slightly modify their state. State drift is undesirable as it can cause wrong evaluations in further computations that may use these devices as input memristors.

Similarly, the design of the NOR4 gate is performed in the same way but involves checking more input cases. Here V_0 is bounded from 1.89 V (for any combination having two inputs at ‘1’, such as “0011”) up to 2.47 V (for the “0000” input case). Interestingly, Fig. 6b shows how the design space of NOR n is modified with increasing fan-in. The range of valid V_0 values results different for different numbers of inputs, first getting wider up to a particular point, and then shrinking.

Once the design space for NOR n was defined, then variability was injected in the involved memristors. The error rate of NOR2 and NOR4 for variability factor $k = 5$ is displayed in Fig. 6c and Fig. 6d, respectively. The error rate is shown separately for each input case (equivalent input cases are grouped in the same category; e.g. “10” and “01”, or “0001” and “0100”, since what is important is just the number of logic ones, rather than the specific bit sequence). Apparently, the most affected case is the “all-zero” input as V_0 increases, since the higher the applied voltage the larger the state drift in the input memristors. Although all V_0 values within the design process should in principle be valid, it turns out that the error rate in some cases is simply unacceptable. The average error rate is minimized for $V_0 = 1.95\text{ V}$ in NOR2 and for $V_0 = 2\text{ V}$ in NOR4, respectively, highlighting that memristor device variability must be taken into account in the design process. Note that in NOR4 the average error rate seems more flat than in NOR2. This, owing to the fact that most of input cases exhibit similar and low error rates, whereas the critical case remains only one (the “all-zero” case) so it weighs less in the average calculation.

Next, the error rate of AOI22 was computed. Since AOI22 was implemented via sequential NOR2 operations, the design space of NOR2 was taken into account and only values around $V_0 = 1.95\text{ V}$ were tested, i.e. around the value that minimized the error rate for NOR2. Figure 6e shows the results for variability factor $k = 5$. The error rate is detailed for each grouped input case (equivalent input cases are grouped in pairs each corresponding to the inputs of one of the two first-level NOR2 gates. So, for example, “00|01” also includes “00|10”, “01|00”, and “10|00”). We observed that the most error-prone cases are the ones with a “00” input pair. This was somehow expected since “00” is also the input combination with highest error rate in NOR2 (Fig. 6b), thus in AOI22 the first-level NOR2 gates generate quite unreliable outputs that are next used as inputs in the second level, to eventually produce an even less reliable final output. Moreover, the average error rate, here minimized for $V_0 = 1.9\text{ V}$, does not change significantly with V_0 . We also observe that the error rate curve, even for the most critical input case, it is not as steep as in previous results. Generally, in agreement with NOR2 and NOR4 functions, it seems that using higher voltage amplitudes results in increasing error rate. However, the error rate of AOI22 shows higher values than the NOR2 or NOR4 operations, reflecting the undesired consequence of chained operations. A logic operation with a minimum error rate of 15.6% (“00|00” input for $V_0 = 1.9\text{ V}$) is clearly not feasible. So, after every NOR2 operation the output should be properly rewritten to minimize the propagated error rate.

2) CNIMP logic gates

For CNIMP, the only design necessary to be performed is that of the basic CNIMP operation which is used to perform any other logic function. After that, the NOR2, NOR4 and AOI22 are simulated injecting the same amount of variability to finally compute the error rate. Contrary to the MAGIC case, in the CNIMP design there are three design parameters: $V_{\text{COND}+}$, $V_{\text{COND}-}$ and R_G . So, in order to keep the design process as simple and short as possible, we reduced the test set of $V_{\text{COND}+}$ and R_G values and rather swept only $V_{\text{COND}-}$. The input voltage pulse width was again kept at 200 ns as in MAGIC.

The corresponding results are shown in Fig. 7a. Different designs were obtained for different R_G values. Then, NOR2, NOR4 and AOI22 operations were tested to obtain the error rate, as shown in Fig. 7b, Fig. 7c and Fig. 7d, respectively. For these two operations, the $R_G = 10\text{ k}\Omega$ and $V_{\text{COND}+} = 1\text{ V}$ were fixed, and $V_{\text{COND}-}$ was tested for -1.1 V, -1.2 V, and -1.3 V. Likewise in the MAGIC case, once again the results show that the error rate is very much dependent on the design parameters. Also, we observe that the CNIMP NOR2 error rate is lower than the MAGIC NOR2 but as the number of inputs increases, the CNIMP error rate results higher than the corresponding MAGIC gate (case NOR4). We even note a set of parameter values that leads to 100% error rate for the “0000” input combination; this clearly being an undesirable effect of state drift, since NOR4 is computed using a chain of four CNIMP operations. Furthermore, the AOI22 results confirm the fact that, although the function is implemented with sequential NOR2 operations, the error rate is not only higher than NOR2 but prohibitive. Generally, in CNIMP-based implementations the chained operations are even more critical than in MAGIC as every NOR n involves a chain of basic

CNIMP operations. Likewise in NOR4, we confirmed that $V_{\text{COND}} = -1.3$ V value produces completely faulty computations for certain input combinations. Unfortunately, it is not easy to establish any correlations between NOR2 and AOI22 faulty input cases as it was for the MAGIC AOI22. However, it is noticeable that input combinations with a high number of logic ‘0’ and those including some logic ‘1’ evaluated in early stages of the computation (for instance, “1000”, “0100” or “1100”) are the ones with highest error rate. Finally, the minimum average error rate was displaced here w.r.t. that in CNIMP NOR2, to $V_{\text{COND}} = -1.1$ V.

All in all, MAGIC and CNIMP were proved to be quite sensitive to variability in memristor states even after being properly designed, and even when considering quite a large memristance window. What is more, since large voltages are generally required, the state drift issue appears as well. Chained operations may worsen the situation increasing the amount of faulty operations to prohibitive levels and thus requiring for extra intermediate steps to restore the memristors states between logic operations. Therefore, it is expected that a logic scheme which does not imply conditional switching of the memristor states would most likely constitute a more viable and robust approach.

3) Ratioed logic gates

Last, we present the results concerning the design exploration of the proposed ratioed-logic scheme. The $V_{\text{DD}} - V_{\text{REF}}$ difference must be low enough to not disturb the state of the input memristors. Using the $V_{\text{read}} = 0.5$ V amplitude as a reference, we used $V_{\text{DD}} = 3.3$ V to operate the 0.35 μm PMOS load transistor and then set $V_{\text{REF}} = 2.8$ V. Moreover, the PMOS channel resistance must be similar to the LRS resistance, which is accomplished with proper sizing and by applying a gate voltage of 1.6 V. The voltage pulse width was just 20 ns, much lower when compared to the pulses applied in the MAGIC and CNIMP cases, since no switching of memristors is required.

Regarding the evaluation of the variability impact, since the error rate of NOR2 is zero, it was meaningless to use the error rate as a metric of evaluation. Alternatively, the distributions of the V_{out} values for each different input combination, was found to be much more illustrative for this logic style. The simulation results are shown in Fig. 8a. Each distribution is colored depending on the variability factor k used in the simulation, i.e. blue for $k = 1$, red for $k = 5$ and black for $k = 10$, whereas the insets show separately the details of each distribution. As expected, in the “00” input case V_{out} is near V_{DD} and it appears with a very narrow distribution. On the other hand, “01/10” and “11” input cases have broader distributions that are found centered in particular values as explained theoretically in Section II. Interestingly, the results confirm that the three cases are clearly distinguishable, and a comparison voltage V_{COMP} can be easily defined to correctly identify the values corresponding to logic ‘1’ or logic ‘0’. The NOR4 results in Fig. 8b show similar performance. Indeed, as there are more input memristors in parallel, the V_{out} is displaced further from V_{DD} . Again, V_{COMP} clearly separates the input cases that give a logic ‘1’ output from those that give a logic ‘0’ output.

Next, Fig. 8c aims to underline the really high tolerance of the proposed scheme to device variability. In fact, the graph shows the V_{out} range evolution with increasing number of inputs of the NOR gate. The output is shown only for the all-zero input case (“00...00”) and for the combinations with only one input storing a logic ‘1’ (shown as “00...01”), since their results are closer to V_{COMP} , as seen in Figs. 8a,b. The green line shows an extrapolation following the results without any variability for input number $n = 2, 4, 8$ and 16, whereas the simulation results shown were taken after the injection of variability. It can be noticed that the two cases remain separated enough so as to be properly distinguished. Eventually, the “00...00” input case theoretically intersects the V_{COMP} line approx. when $n = 64$! However, such limitation could be erased by justifying accordingly the V_{COMP} level to the number of inputs. Hence, only when the V_{out} values for these two extreme input combination cases are too close, the error rate will be non-zero.

Here we did not include any AOI22 simulation results since the performance of the basic operation, i.e. NOR_n , was found a priori superior and practically error-free. On the other hand, since output here is expressed in voltage instead of memristance, conducting chained operations in this scheme assumes adding an intermediate step to store the logic output to a memristor state which will later act as input variable. Therefore, in principle delay of computations following the proposed ratioed scheme would be more. However, as commented previously, unless an intermediate “regenerating” step is included both in MAGIC and CNIMP gates, chained operations in either of these styles is unacceptably error-prone. So, after all, requiring an extra step to store intermediate results in rationed logic operations is not that much a disadvantage, considering the high gains in reliability and also the much shorter computation time which involves only sensing the memristor states.

As we mentioned earlier, Scouting Logic was another alternative that performs variability-tolerant logic operations. It is also a switchless logic style, hence sharing some advantages with our memristor-based ratioed logic. Nevertheless, our proposal proves robustness with a straight-forward, non-iterative, design

process and the scalability on number of inputs in, for instance, the NOR operation, has been guaranteed.

VI. CONCLUSIONS

The paper compares the performance of three memristive logic styles in terms of tolerance to device variability. Simulation results, based on realistic models for all devices, confirm that a variability-aware design is required in the assessment of memristive logic design schemes. The analysis shows that the behavior of MAGIC and CNIMP is highly sensitive to design parameters, increasing significantly the error rate up to unacceptable levels. State drift is also observed in the design process and highly impacts chained operations. On the other hand, the proposed ratioed logic scheme with a much simpler design space exploration, was proven much more robust against device variability. Also, it is crossbar-compatible, fast, and it does not affect the device endurance as computations do not involve switching the memristor states. Smaller required amplitudes are expected to result in lower consumption as well (not covered in this work) and certainly avoid state drift. Everything considered, it distinguishes as a good candidate for in/near-memory logic in future computing systems.

REFERENCES

- [1] L. O. Chua, "Memristor - The Missing Circuit Element," *IEEE Trans. Circuit Theory*, vol. 18, no. 5, pp. 507-519, 1971
- [2] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, pp. 80 - 83, 2008
- [3] "International Technology Roadmap for Semiconductors (ITRS)," 2013. [Online]. Available: <http://www.itrs.net/>. [Accessed June 2014]
- [4] H.-S. P. Wong, *et al.*, "Metal-Oxide RRAM," *IEEE Proc.*, vol. 100, no. 6, pp. 1951-1970, 2012
- [5] I. Vourkas and G. Ch. Sirakoulis, "Emerging Memristor-based Logic Circuit Design Approaches: A Review," *IEEE Circ. and Syst. Mag.*, vol. 16, no. 3 (3rd quarter), pp. 15-30, 2016
- [6] H. Li, T. F. Wu, S. Mitra, and H.-S. P. Wong, "Resistive RAM-centric computing: Design and modeling methodology," *IEEE Trans. Circ. Syst. I, Reg. Papers*, vol. 64, no. 9, pp. 2263-2273, 2017
- [7] M. Prezioso *et al.*, "Training and operation of an integrated neuromorphic network based on metaloxide memristors," *Nature*, vol. 521, no. 7550, pp. 61-64, 2015
- [8] I. Vourkas, A. Batsos, and G. Ch. Sirakoulis, "SPICE modeling of nonlinear memristive behavior," *Int. J. Circ. Theor. Appl.*, vol. 43, no. 5, pp. 553-565, 2015
- [9] C. Yakopcic, T. M. Taha, G. Subramanyam, and R. E. Pino, "Memristor SPICE Modeling," in *Advances in Neuromorphic Memristor Science and Applications*, 1st Ed. The Netherlands: Springer, 2012, pp. 211-244
- [10] Y. Pershin and M. Di Ventra "SPICE Model of Memristive Devices with Threshold," *Radioengineering*, vol. 22, no. 2, pp. 485-489, 2013
- [11] J. Zha, H. Huang, and Y. Liu, "A Novel Window Function for Memristor Model with Application in Programming Analog Circuits," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 63, no. 5, pp. 423-427, 2016
- [12] V. Ntinis, I. Vourkas, G. Ch. Sirakoulis, and A. Adamatzky, "Modeling Physarum space exploration using memristors," *J. Phys. D: Appl. Phys.*, vol. 50 (174004), 2017
- [13] M. A. Nugent, T. W. Molter, "Thermodynamic-RAM technology stack," *Int. Journal of Parallel, Emergent and Distributed Systems*, vol. 33, no. 4, pp. 430-444, 2018
- [14] Z. Jiang, Y. Wu, S. Yu, L. Yang, K. Song, Z. Karim, and H.-S. Wong, "A Compact Model for Metal-Oxide Resistive Random Access Memory with Experiment Verification," *IEEE Trans. Electron Devices*, vol. 63, no. 5, pp. 1884-1892, 2016
- [15] J. Borghetti *et al.*, "Memristive switches enable 'stateful' logic operations via material implication," *Nature*, vol. 464, pp. 873-876, Apr. 2010
- [16] S. Kvaterny, *et al.*, "MAGIC—Memristor-Aided Logic," *IEEE Trans. Circ. Syst. – II: Expr. Briefs*, vol. 61, no. 11, pp. 895 - 899, 2014
- [17] Q. Chen, X. Wang, H. Wan, and R. Yang, "A Logic Circuit Design for Perfecting Memristor-Based Material Implication," *IEEE Trans. Computer-Aided Design of Integ. Circ. Syst.*, vol. 36, no. 2, pp. 279 - 284, 2017
- [18] S. Kvaterny, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MRL - Memristor Ratioed Logic," *Proc. Int. Workshop on Cellular Nanoscale Networks and their Appl.*, pp. 1-6, August 2012
- [19] L. Guckert, E. E. Swartzlander, "MAD Gates—Memristor Logic Design Using Driver Circuitry," *IEEE Trans. Circ. Syst. – II: Expr. Briefs*, vol. 64, no. 2, pp. 171-175, 2017
- [20] J. Reuben, *et al.*, "Memristive logic: A framework for evaluation and comparison," *27th Int. Symp. Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Thessaloniki, Greece, 25-27 Sept. 2017
- [21] G. Papandroulidakis, I. Vourkas, A. Abusleme, G. Ch. Sirakoulis, A. Rubio, "Crossbar-Based Memristive Logic-in-Memory Architecture," *IEEE Trans. Nanotechnol.*, vol. 16, no. 3, pp. 491-501, 2017
- [22] L. Xie, *et al.*, "Scouting Logic: A Novel Memristor-Based Logic Design for Resistive Computing," *2017 IEEE Computer Society Annual Symp. VLSI (ISVLSI)*, Bochum, Germany, 3-5 July 2017
- [23] Z. Jiang, H. Li, J. H. Engel, S. Yu, and X. Guan, "Stanford-PKU RRAM Model v2.0.0," [Online]. Available: <https://nano.stanford.edu/stanford-rram-model> [Accessed October 2018]
- [24] M. Escudero, I. Vourkas, A. Rubio, F. Moll, "On the Variability-aware Design of Memristor-based Logic Circuits," *18th IEEE Int. Conf. on Nanotechnol. (IEEE-NANO)*, Cork, Ireland, 23-26 July, 2018
- [25] M. Escudero, I. Vourkas, A. Rubio, F. Moll, "Variability-Tolerant Memristor-based Ratioed Logic in Crossbar Array," *2018 Int. Symp. on Nanoscale Architectures (NANOARCH)*, Athens, Greece, 18-19 July, 2018
- [26] I. Vourkas, G. Ch. Sirakoulis, "A Novel Design and Modeling Paradigm for Memristor-Based Crossbar Circuits," *IEEE Trans. Nanotechnol.*, vol. 11, no. 6, pp. 1151 - 1159, 2012
- [27] I. Vourkas, M. Escudero, G. Papandroulidakis, G. Ch. Sirakoulis, and A. Rubio, "Variability Challenges in Emerging Memristor-based Logic Circuits," *HiPEAC'17 Workshop on Memristor Technology, Design, Automation and Computing (mDAC2017)*, Stockholm, Sweden, January 23, 2017

- [28] M. Laiho, E. Lehtonen, "Cellular nanoscale network cell with memristors for local implication logic and synapses," *2010 IEEE Int. Symp. Circ. Syst. (ISCAS)*, Paris, France, 30 May – 2 June, 2010
- [29] M. Maestro-Izquierdo, *et.al.*, "Experimental Time Evolution Study of the HfO₂-Based IMPLY Gate Operation," *IEEE Trans. Electron Devices*, vol. 65, no. 2, pp. 404-410, 2018
- [30] I. Vourkas and G. C. Sirakoulis, "High-Radix Arithmetic-Logic Unit (ALU) Based on Memristors," in *Memristor-Based Nanoelectronic Computing Circuits Architectures*, 1st ed. Cham, Switzerland: Springer, 2015, pp. 149–172
- [31] C. Li, *et al.* "Analogue signal and image processing with large memristor crossbar," *Nature Electronics*, vol. 1, no. 1, pp. 52-59, 2018
- [32] B. Chen, *et al.* "Highly Compact (4F2) and Well Behaved Nano-Pillar Transistor Controlled Resistive Switching Cell for Neuromorphic System Application," *Sci. Rep.*, vol. 4, no. 6863, 2014
- [33] P. L. Thangkhiew, K. Datta, "Scalable in-memory mapping of Boolean functions in memristive crossbar array using simulated annealing," *Journal of Systems Architecture*, vol. 89, pp. 49–59, 2018
- [34] D. N. Yadav, P. L. Thangkhiew, K. Datta, "Look-ahead mapping of Boolean functions in memristive crossbar array," *Integration, the VLSI Journal*, vol. 64, pp. 152–162, 2019

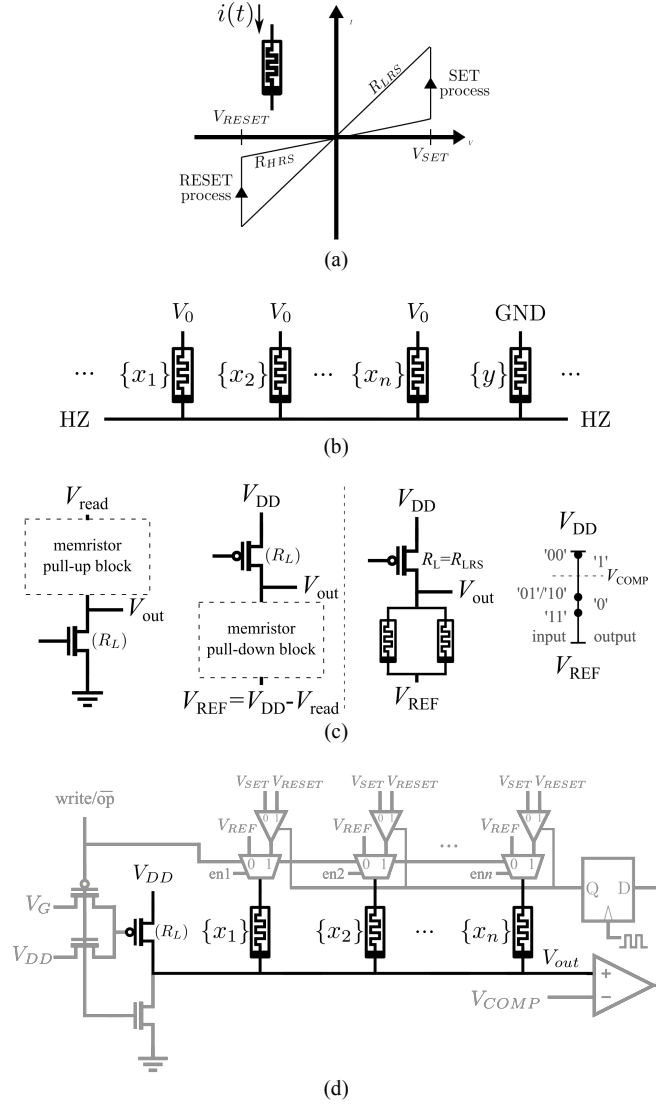


Fig. 1. NOR_n logic gate implementations in a single crossbar row for different logic styles. (a) MAGIC NOR_n operation. (b) CNIMP NOR_n operation performed in three steps. (c) Proposed memristor-based ratioed logic schemes (at left) and NOR2 example (at right). (d) Proposed memristor-based ratioed logic NOR_n operation, including a simplified peripheral circuitry to write back the output result.

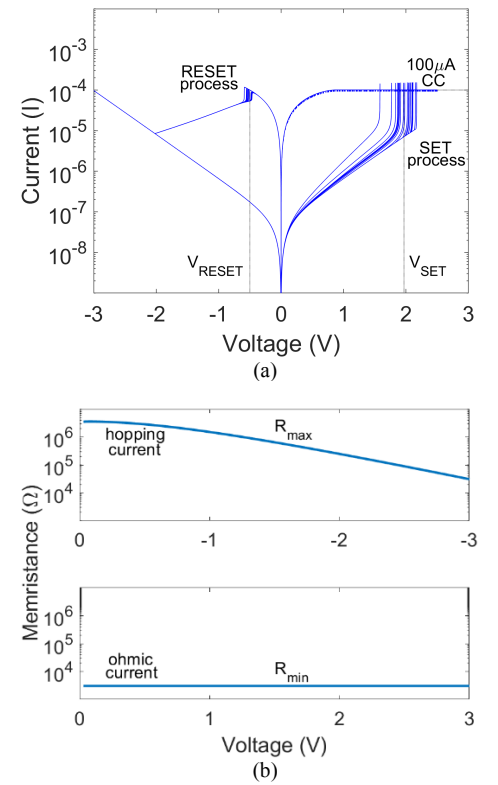


Fig. 3. (a) i - v simulation results concerning 20 cycles with the default variability applied, using a 400 μs -period triangular voltage pulse from -3 V to 2.5 V and a 100 μA compliance current for the SET process. (b) Boundary states R_{MAX} and R_{MIN} as a function of the applied voltage across the memristor (without variability), illustrating the conduction modes in HRS and LRS states.

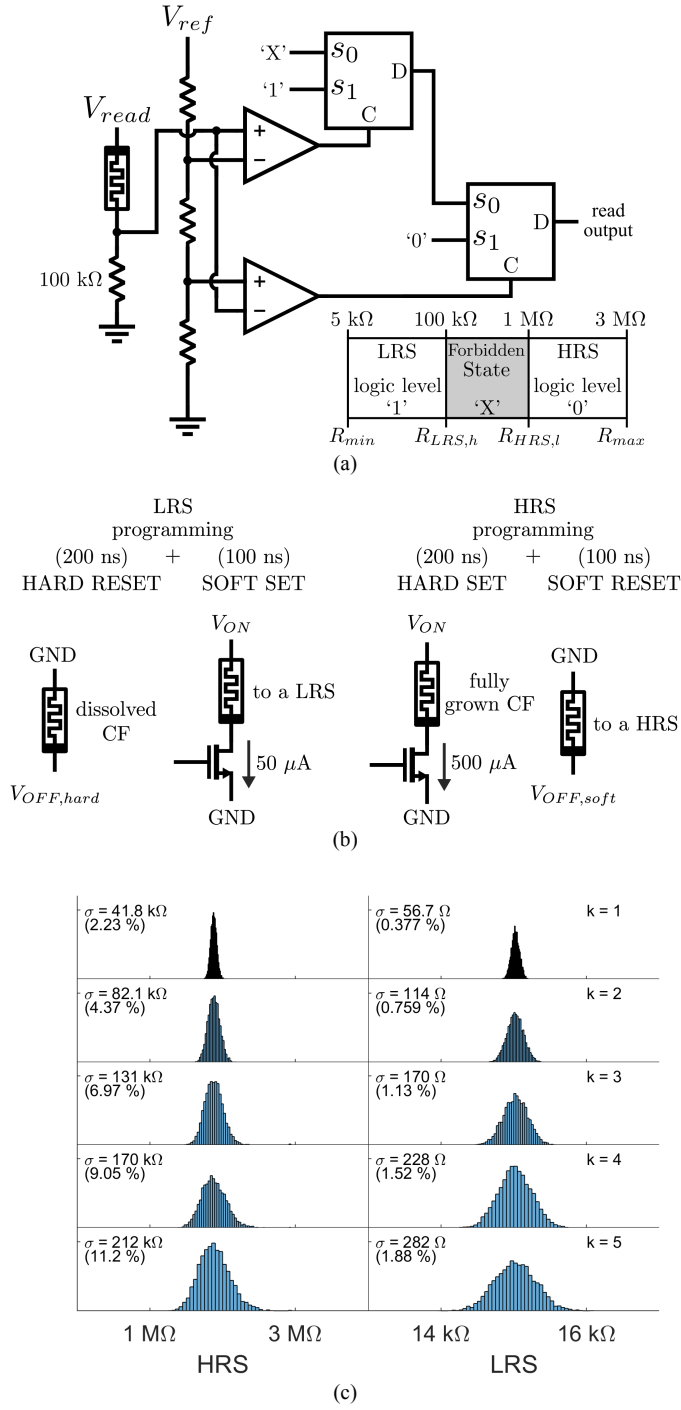


Fig. 4. (a) Memristance binary coding and the reading circuit that senses the memristance in voltage mode and compares with two different reference voltages to decode the memristance as logic '0', '1' or undefined (badly written) 'X'. (b) Two-step programming details for RESET and SET processes. (c) 5000 total HRS (left column) and LRS (right column) reading distributions for different variability factor k values.

Input	Req	Vout	Output
"00"	$R_{HRS}/2$	V_{DD}	'1'
"01"/"10"	$\approx R_{LRS}$	$V_{REF} + (V_{DD} - V_{REF})/2$	'0'
"11"	R_{LRS}	$V_{REF} + (V_{DD} - V_{REF})/3$	'0'

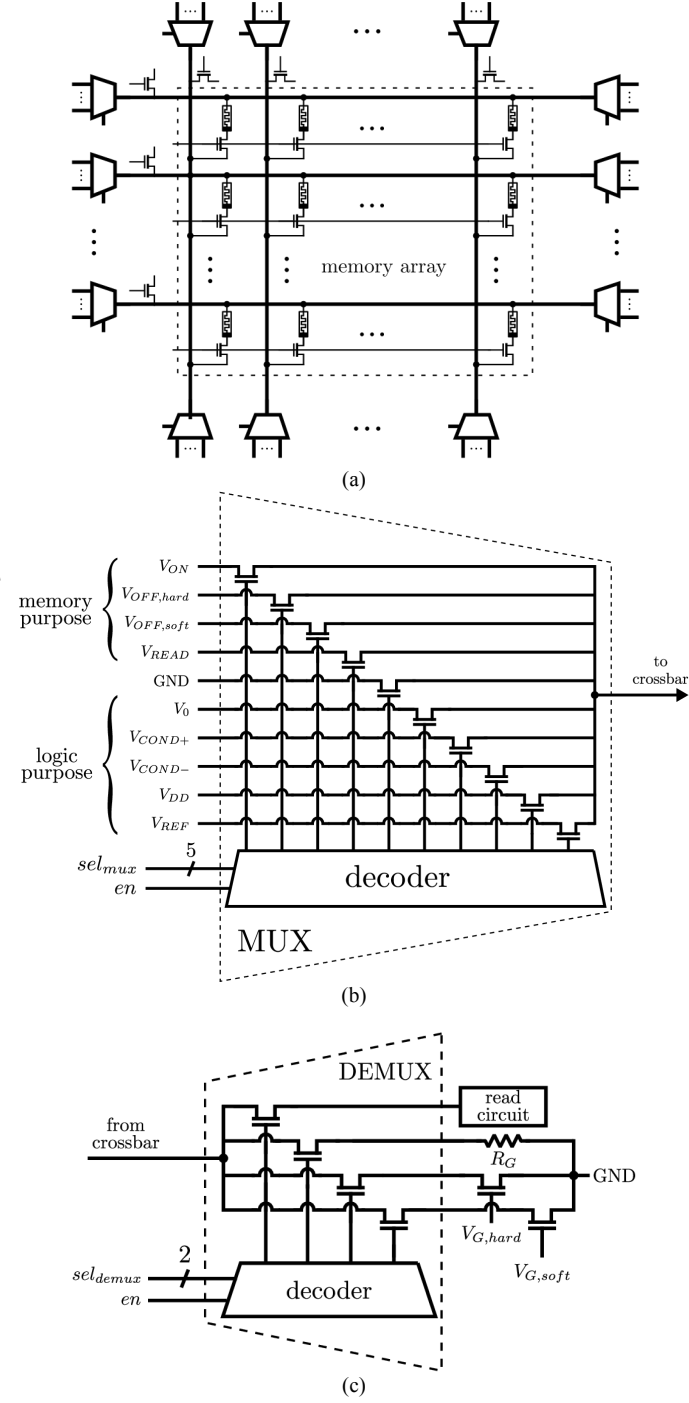
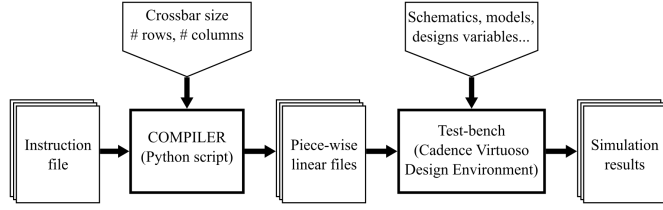


Fig. 5. Simulated system overview. (a) The crossbar memory array and its peripheral circuitry for the requirements of this work. (b) Detailed schematic of the voltage multiplexer unit. (c) Detailed schematic of the demultiplexer unit, providing all the necessary connections needed both for memory and logic operations.



(a)

```

SET 0 0           //LRS programming of device in row 0 and column 0
RESET 0 1        //HRS programming of device in row 0 and column 1
SET 0 2           //LRS programming of device in row 0 and column “
READ r 0          //Read content of row 0
MNOR 2 r 0 0 1 2 //Perform MAGIC NOR2 in row 0, with inputs in
                    //columns 0 and 1, output in column 2
READ r 0          //Read content of row 0

```

(b)

Fig. 6. (a) Description of the simulation flow. (b) Example of an instruction file for a MAGIC NOR2 operation with previously programmed inputs in “01”, where the sequence of operations is shown on the left and some comments about the details of each operation, are given on the right side.

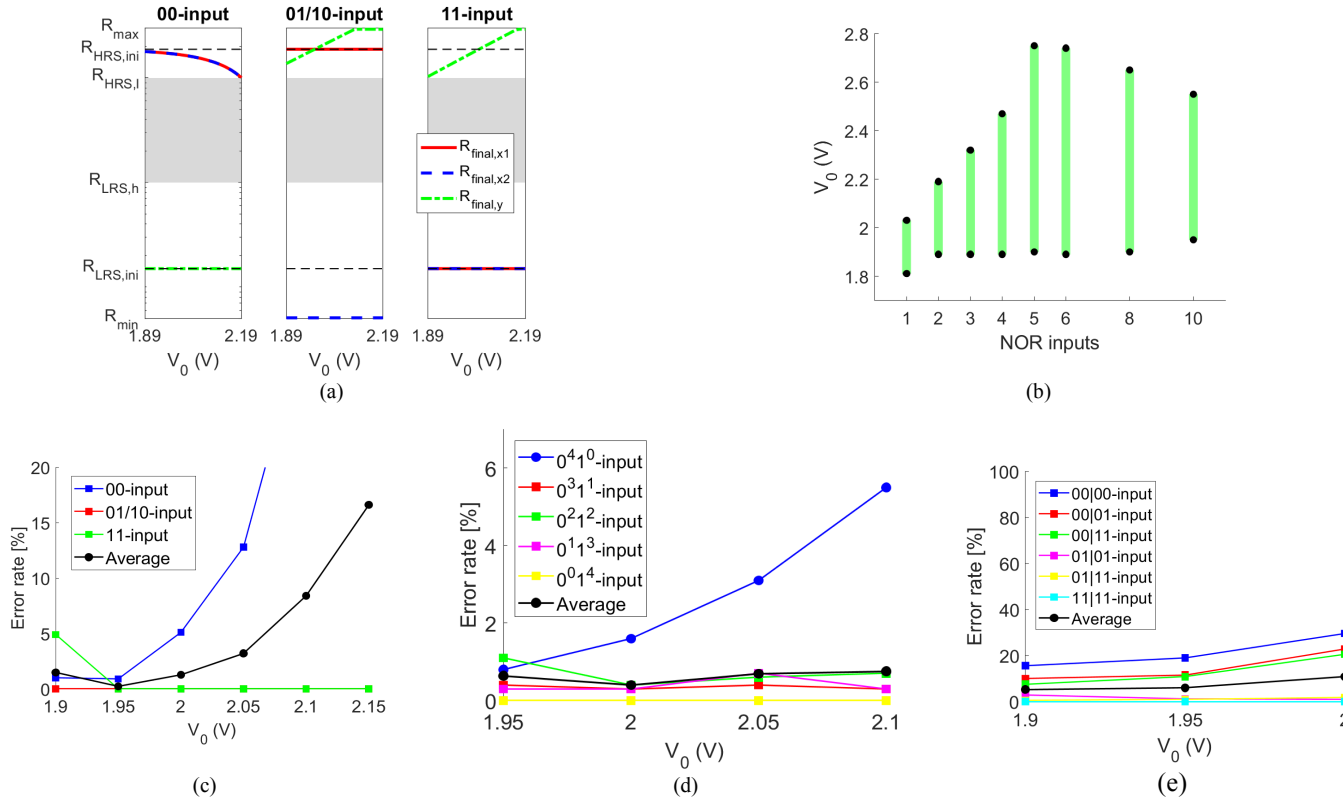


Fig. 7. Simulation results for MAGIC logic style. (a) Design space of NOR2 concerning all possible input combinations. (b) Design space for NOR n for different number of inputs. (c) Error rate of NOR2 after the evaluation of 1000 operations for each possible input combination and $V_0 = \{1.9, 1.95, 2, 2.05, 2.1, 2.15\}$ V. (d) Error rate of NOR4 for each possible combination shown grouped in the form 0^x1^y ; i.e. combinations that involve x zeros and y ones in their binary representation and $V_0 = \{1.95, 2, 2.05, 2.1\}$ V. (e) Error rate of AOI22 evaluated over 1000 operations for every possible input combination grouped as two-input pairs of NOR2. For input combinations of interest, these being 00, 01/10 and 11, for $V_0 = \{1.9, 1.95, 2\}$ V.

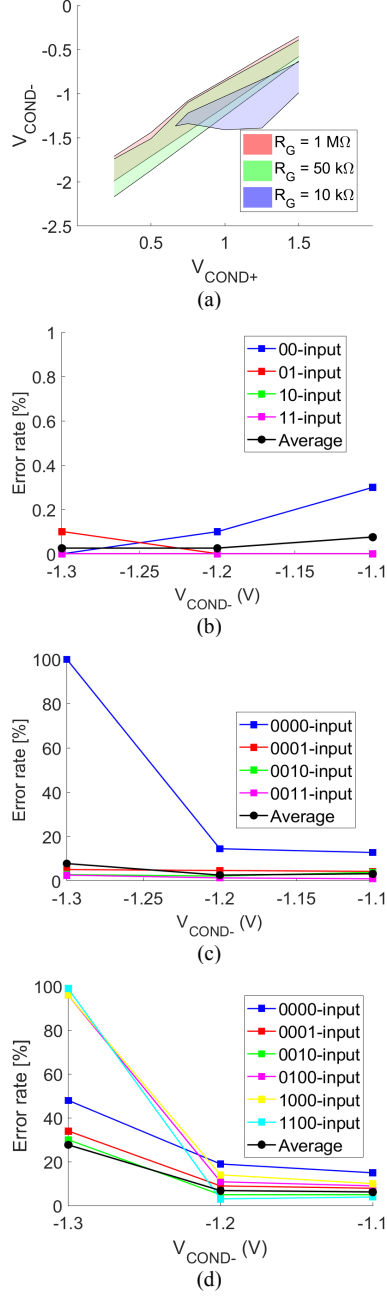


Fig. 8. Simulation results for the CNIMP logic style. (a) Design space of CNIMP operation using a set of R_G values of $\{10, 50, 100\}$ k Ω . (b) Error rate of NOR2 for $R_G = 10 \text{ k}\Omega$ and $V_{COND+} = 1 \text{ V}$, each input combination evaluated 1000 times for each V_{COND-} . (c) Error rate of NOR4 for $R_G = 10 \text{ k}\Omega$ and $V_{COND+} = 1 \text{ V}$, each input combination evaluated 1000 times for each V_{COND-} . (showing only the input cases with higher error rates). (d) Error rate of AOI22 for $R_G = 10 \text{ k}\Omega$ and $V_{COND+} = 1 \text{ V}$, showing only the input cases with higher error rates.

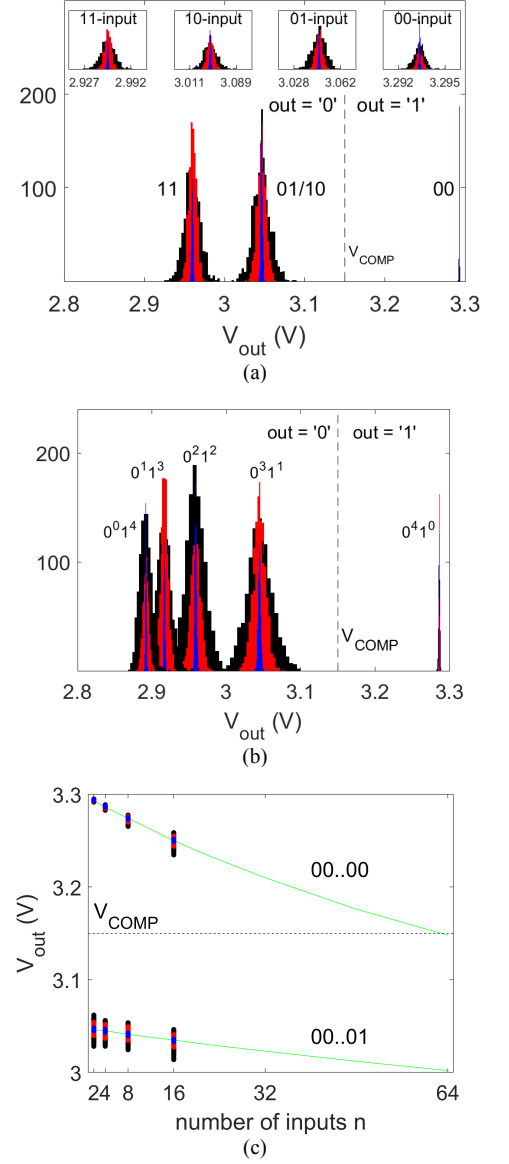


Fig. 9. Simulation results for the ratioed logic style. (a) Histograms of V_{out} for NOR2 after the evaluation of 1000 operations for each possible input combination when $k = 1$ (blue), $k = 5$ (red) and $k = 10$ (black). (b) Histograms of V_{out} for NOR4 after evaluating 1000 operations for each possible input combination for $k = 1, 5$ and 10 . (c) Maximum fan in exploration for the NOR operation, using $k = 1, 5$ and 10 (1000 evaluations for each distribution). The results concern the all-zeros input and those combinations having only one input at logic '1'. The green line tracks V_{out} when $k = 0$.

CROSS-POINT AREA REQUIREMENT						
Logic Style	Input position	NOR			OR	
		IN	AUX	OUT	IN	AUX
MAGIC	Not aligned	2	2	1	2	3
	Aligned	2	0	1	2	1
CNIMP	Not aligned	2	2	1	2	3
	Aligned	2	0	1	2	1
Ratioed Logic	Not aligned	2	1	1	2	1
	Aligned	2	0	1	2	0